Sandia National Laboratories

Exceptional service in the national interest

ME469: Common Discretization Approaches: Control-volume Finite Element Method (CVFEM)

Stefan P. Domino[1,2]

[1] Computational Thermal and Fluid Mechanics, Sandia National Laboratories

[2] Institute for Computational and Mathematical Engineering, Stanford

SAND2018-4536 PE

Review: Implicit vs Explicit

$$\frac{\partial \phi}{\partial t} + v\frac{\partial \phi}{\partial x} = \nu\frac{\partial^2 \phi}{\partial x^2}$$

- Recall, *forward-in-time* and *central-in-space* derivatives: FT-CS ***NOT STABLE***

$$\frac{\phi_j^{n+1} - \phi_j^n}{\Delta t} + v\frac{\phi_{j+1}^n - \phi_{j-1}^n}{2\Delta x} = \nu\frac{\phi_{j+1}^n - 2\phi_j^n + \phi_{j-1}^n}{\Delta x^2}$$

$$\phi_j^{n+1} = F\left(\phi_j^n, \phi_{j+1}^n, \phi_{j-1}^n, v, \nu, \Delta t, \Delta x\right)$$

- And, *backward-in-time* and *central-in-space* derivatives: BT-CS ***STABLE, however, oscilliatory at high Peclet numbers***

$$\frac{\phi_j^{n+1} - \phi_j^n}{\Delta t} + v\frac{\phi_{j+1}^{n+1} - \phi_{j-1}^{n+1}}{2\Delta x} = \nu\frac{\phi_{j+1}^{n+1} - 2\phi_j^{n+1} + \phi_{j-1}^{n+1}}{\Delta x^2}$$

$$\phi_j^{n+1} = F\left(\phi_j^{n+1}, \phi_{j+1}^{n+1}, \phi_{j-1}^{n+1}, v, \nu, \Delta t, \Delta x\right) \qquad A\phi^{n+1} = b$$

Matrix Assembly, Compact form

Defining a stationary advection and diffusion system for scalar ϕ with constant properties and positive velocity as

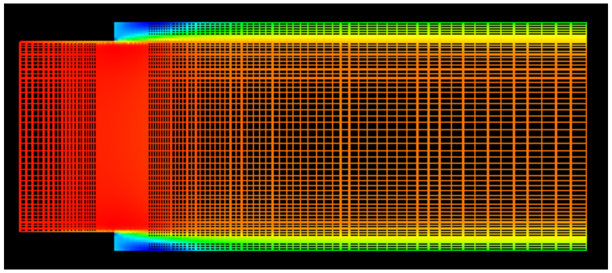
$$\frac{\partial \rho u_j \phi}{\partial x_j} - \frac{\partial}{\partial x_j}\left(\rho D \frac{\partial \phi}{\partial x_j}\right) = 0, \tag{7.20}$$

and, by utilizing the advection and diffusion operators outlined in Section 7.2, yields the following one-dimensional matrix form for solution i defined in Fig. 7.6:

$$\left(\frac{\rho u}{2}\begin{bmatrix}-1 & 0 & 1\end{bmatrix} + \frac{\rho D}{\Delta x}\begin{bmatrix}-1 & 2 & -1\end{bmatrix}\right)\begin{bmatrix}\phi_{i-1} \\ \phi_i \\ \phi_{i+1}\end{bmatrix}. \tag{7.21}$$

The matrix system with coefficients $a_{i,j}$ can be written as

$$a_{i,i}\phi_i = a_{i,i-1}\phi_{i-1} + a_{i,i+1}\phi_{i+1}, \tag{7.22}$$



Each DOF location will have a dedicated matrix "row" and set of Columns defined by its connectivity

Matrix Assembly, Re-visited: Central Advection

- https://github.com/NaluCFD/Nalu/blob/master/src/AssembleScalarEdgeSolverAlgorithm.C

```
//====================================
// advective flux
//====================================

// 2nd order central
const double qIp = 0.5*( qNp1L + qNp1R );


// central; left; collect terms on alpha and alphaUpw
alhsfac = 0.5*tmdot*(pecfac*om_alphaUpw + om_pecfac*om_alpha);
p_lhs[0] += alhsfac;
p_lhs[1] += alhsfac;
// central; right; collect terms on alpha and alphaUpw
p_lhs[2] -= alhsfac;
p_lhs[3] -= alhsfac;

// total flux left
p_rhs[0] -= aflux;
// total flux right
p_rhs[1] += aflux;
```

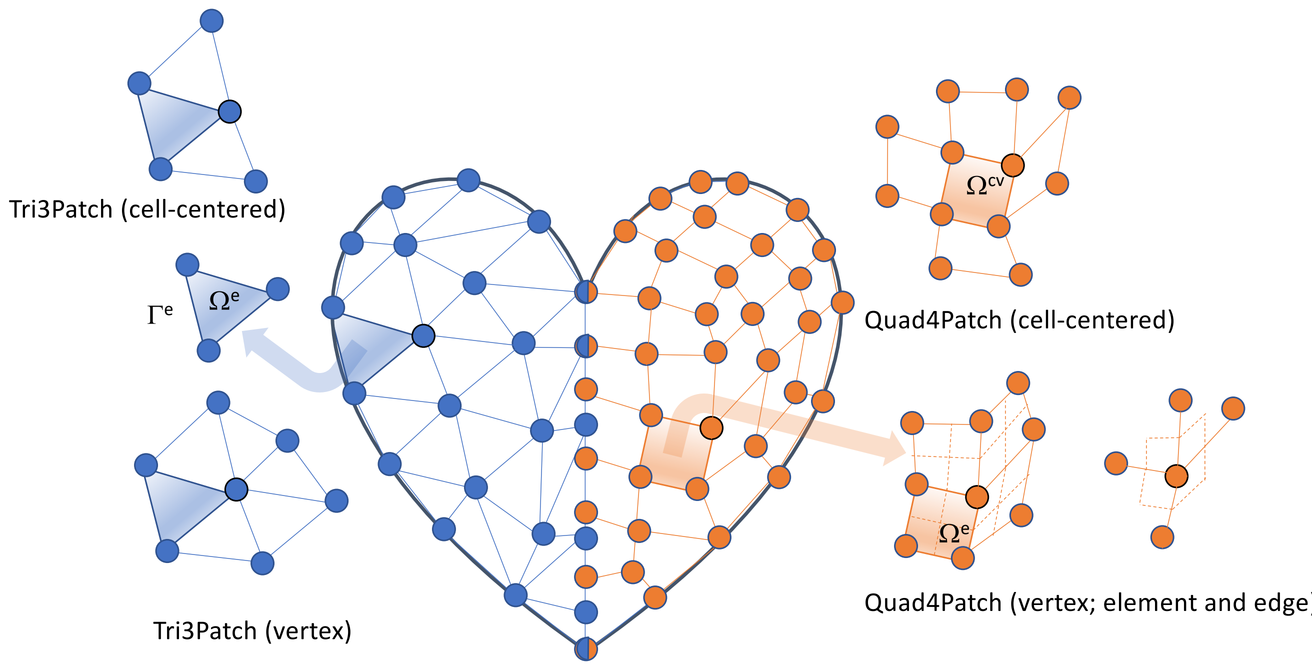
Matrix Assembly, Re-visited: Diffusion

- https://github.com/NaluCFD/Nalu/blob/master/src/AssembleScalarEdgeDiffSolverAlgorithm.C

```
//======================================
// diffusive flux
//======================================
double lhsfac = -viscIp*asq*inv_axdx;
double diffFlux = lhsfac*(qNp1R - qNp1L) + nonOrth*nocFac;

// first left
p_lhs[0] = -lhsfac;
p_lhs[1] = +lhsfac;
p_rhs[0] = -diffFlux;

// now right
p_lhs[2] = +lhsfac;
p_lhs[3] = -lhsfac;
p_rhs[1] = diffFlux;
```

Review of Discretization Options: New, a nodal-basis…

- Degree-of-freedom (DOF) for:
  - Cell-centered: Stencil is based on a element:face:element
  - DOFs at vertices of elements, or “nodes”, element:node (CVFEM, FEM), edge:node (EBVC)



- Definition of an interpolation function:

$$\phi_{ip} = \sum_n N_n^{ip} \phi_n$$

- N_n^{ip} is the Lagrange function associated with node n
- ϕ_n is the value of the DOF at node n
- The nodal basis functions obey equipartition of unity and satisfy, $N_n^{x_j} = \delta_{nj}$

Fundamentals of Discretization: Surface vs Volume Integrations

- Given a partial differential equation (PDE) and associated volumetric form:

$$\int_{\Omega} \frac{\partial F_j}{\partial x_j} d\Omega = \int_{\Omega} S d\Omega$$

- Applying Gauss Divergence provides the standard finite volume form for fluxes in surface integral form (no distinction between internal control volume faces and boundary faces):

$$\sum_k \int_{\Omega_k} \frac{\partial F_j}{\partial x_j} d\Omega_k = \sum_k \int_{\Omega_k} S d\Omega_k \longrightarrow \sum_k \int_{\Gamma_k} F_j n_j d\Gamma_k = \sum_k \int_{\Omega_k} S d\Omega \longrightarrow \int F_j n_j dS = \int S dV$$

- We can also multiple PDE by an arbitrary test function, w, and integrate over a volume,

$$\int w \frac{\partial F_j}{\partial x_j} dV = \int w S dV$$

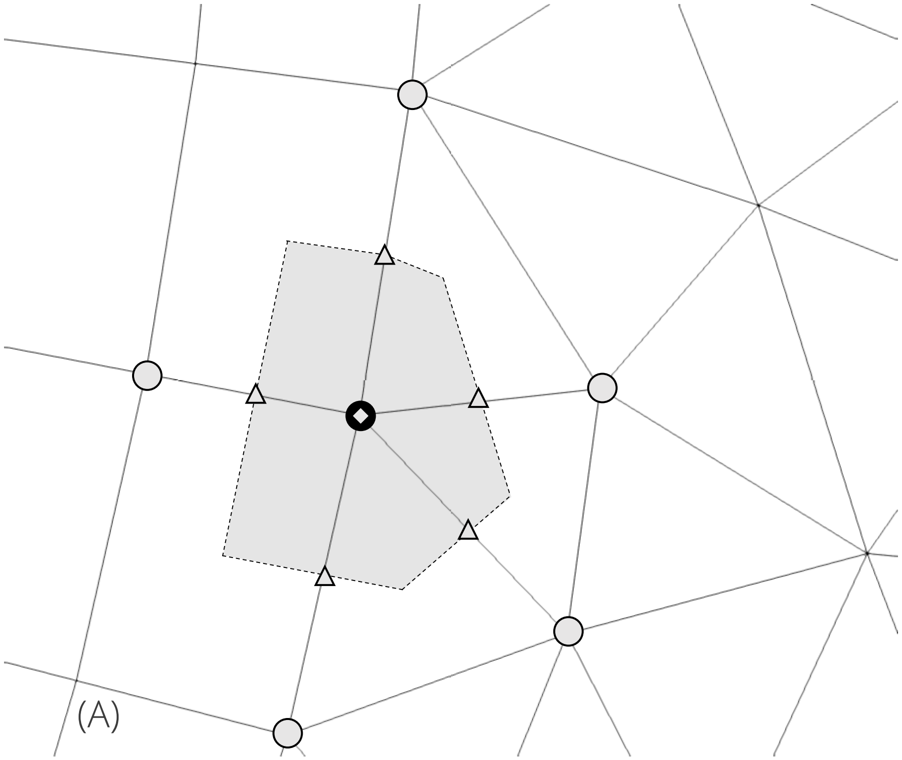
Next, integrate by parts and apply Gauss-Divergence. Note, that test function must be differentiable – shown here, at least once..

$$-\int F_j \frac{\partial w}{\partial x_j} dV + \int \frac{\partial w F_j}{\partial x_j} dV = -\underset{\uparrow\ \text{Interior}}{\int} F_j \frac{\partial w}{\partial x_j} dV + \underset{\uparrow\ \text{Boundary}}{\int} w F_j n_j dS$$

$$\frac{\partial w F_j}{\partial x_j} = w \frac{\partial F_j}{\partial x_j} + \frac{\partial w}{\partial x_j} F_j$$

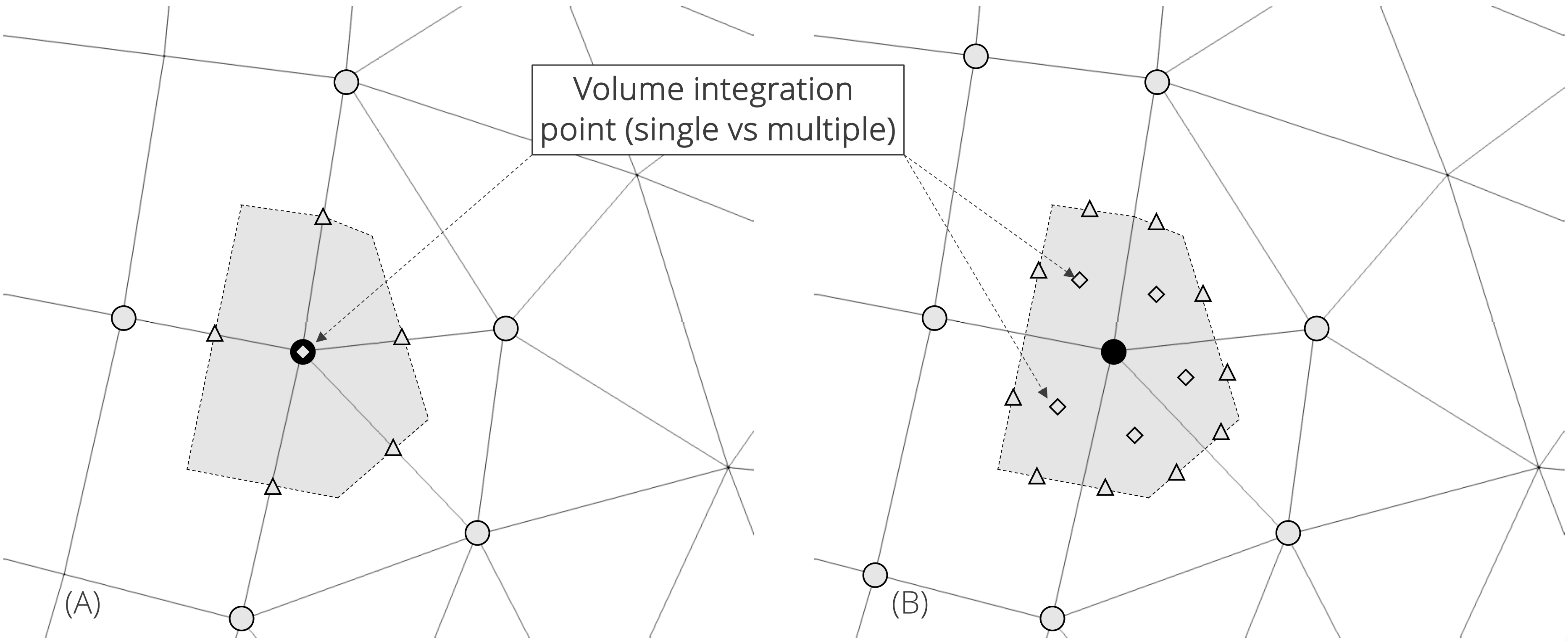
Edge-Based Vertex-Centered Leverages the Dual-Volume Element-based Description of a Control Volume Finite Element Method (CVFEM)

- EBVC (A) and CVFEM (B) – As shown below, the dual-volume and integration point layout is very similar



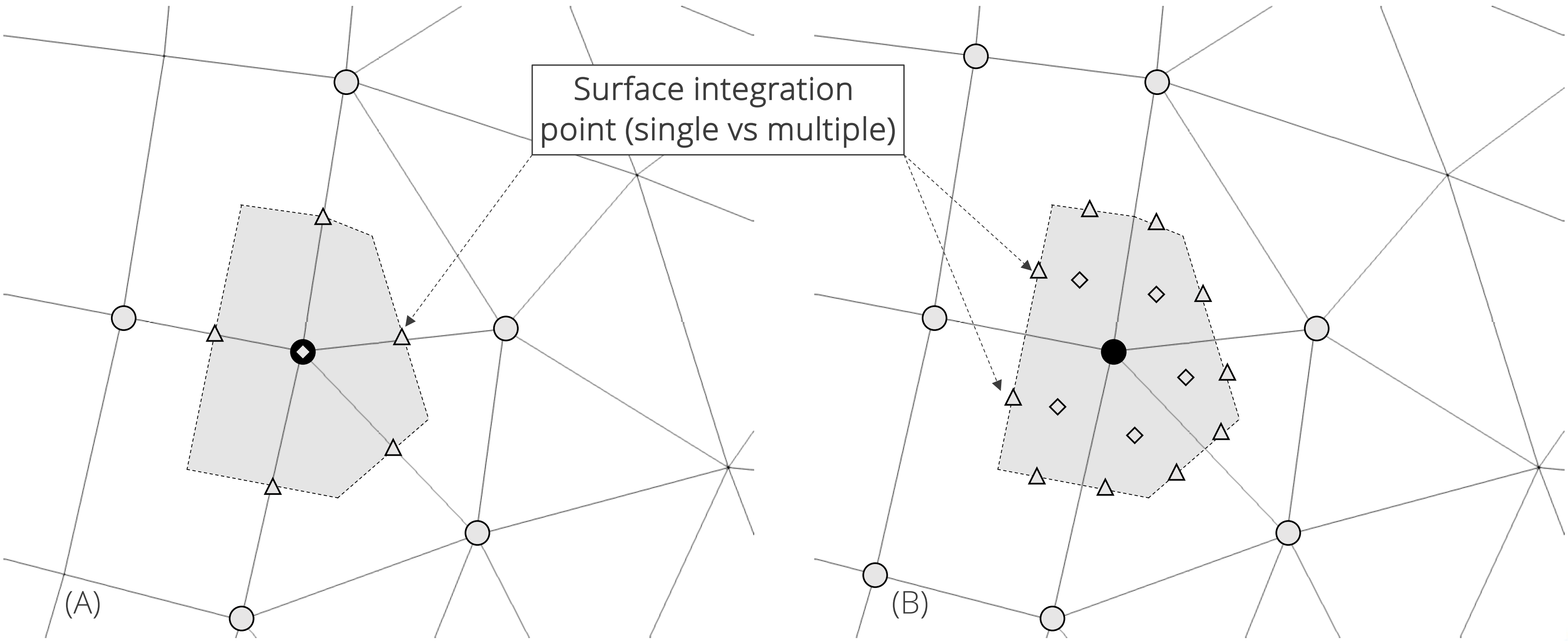
Edge-Based Vertex-Centered Leverages the Dual-Volume Element-based Description of a Control Volume Finite Element Method (CVFEM)

- EBVC (A) and CVFEM (B) – As shown below, the dual-volume and integration point layout is very similar



Edge-Based Vertex-Centered Leverages the Dual-Volume Element-based Description of a Control Volume Finite Element Method (CVFEM)

- EBVC (A) and CVFEM (B) – As shown below, the dual-volume and integration point layout is very similar



Deep Dive on CVFEM

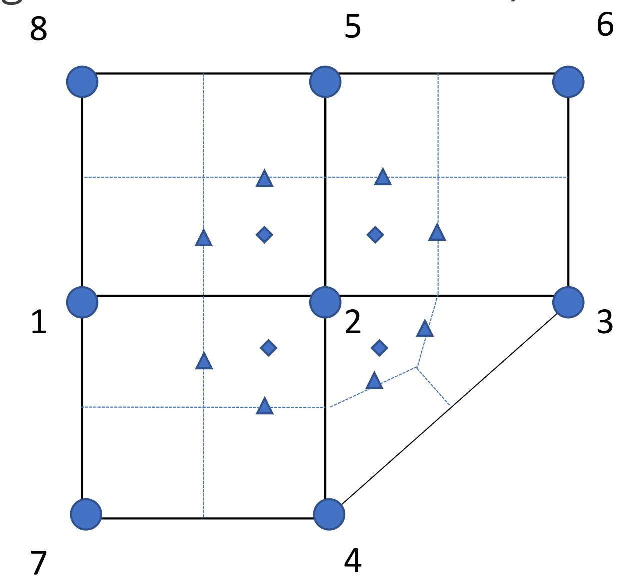
- CVFEM is a discretization scheme that:
  - Iterates over locally-owned elements for Time/Source/etc. (volumetric-based terms)
  - Iterates over locally-owned elements for Advection/Diffusion/etc. (integrated by parts terms)

Below is the patch of elements connected to node 2 (a global matrix row number)

- A *dual-volume* is defined within each element

Any value of the DOF within the element can be computed based on:

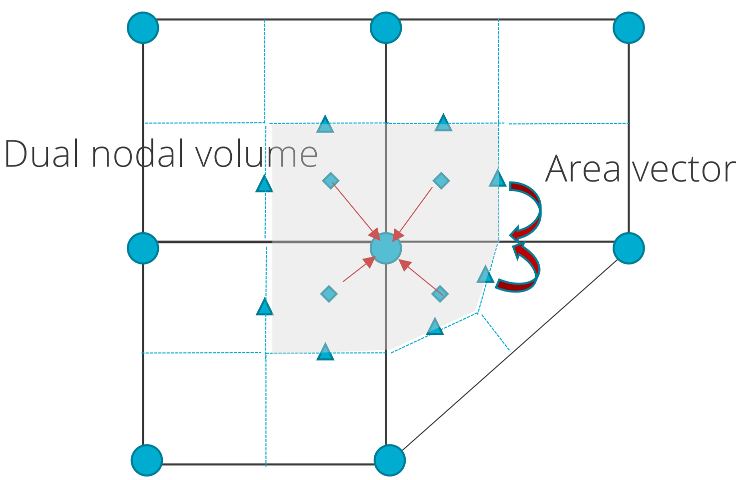
$$\phi_{ip} = \sum_n N_n^{ip} \phi_n$$



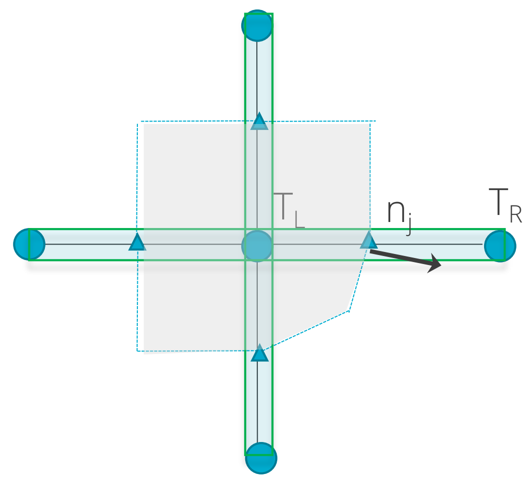
Sample patch of elements

The Control Volume for EBVC is Defined by the ***Dual-Volume***

- All primitives are collocated at the vertices of the elements with equal-order interpolation
- A dual mesh is constructed to obtain flux and volume quadrature locations
- Classic two-state, “L” and “R” approach provides spatially second-order accuracy
- Iterate ***Nodes*** for volume-based contributions
- Iterate ***Edges*** for surface-based contributions

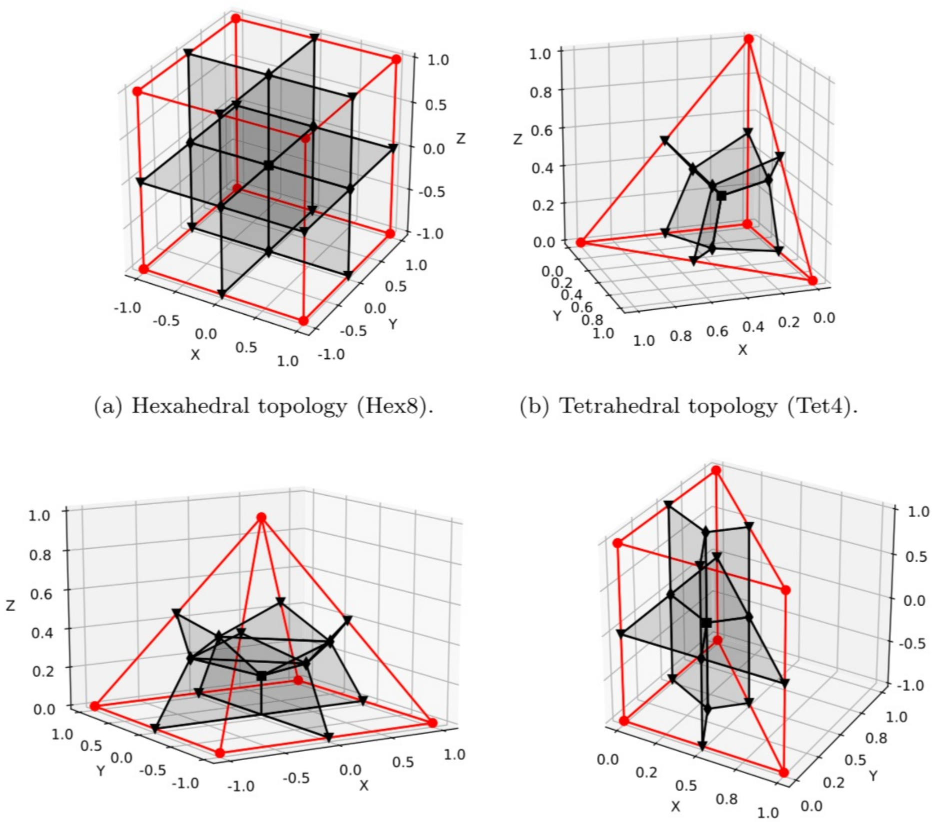


Dual-volume definition



Edge-based stencil

Dual Volume Definitions for Hybrid (Hex/Tet/Pyramid/Wedge) Meshes



(a) Hexahedral topology (Hex8). (b) Tetrahedral topology (Tet4).

(c) Pyramid topology (Pyramid5). (d) Wedge topology (Wedge6).

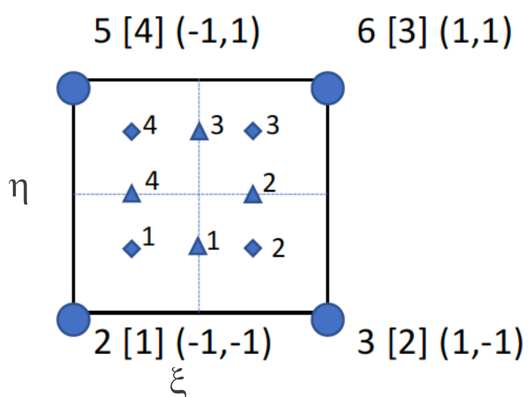
Fig. 1. CVFEM element and dual-volume definition for the low-order topologies.

- Domino, et. al, “An assessment of atypical mesh topologies for low-Mach large-eddy simulation” 2019
- Generalized unstructured meshes support the ability to capture complex geometries, while minimizing the meshing time
- For example, near a solid wall, you might have a near-structured Hex-based mesh that is transitioned to unstructured
- For the mesh to be conformal, the faces for each adjacent element topology should match
- Drives Hex8 (Quad4 face) : Pyramid5 (Quad4 face); Pyramid5 (Tri3 face) to Tet4 (Tri3 face)
  - The Pyramid is known as a *transition* element

Deep Dive on CVFEM: Element-loops with Rich Basis

- For each element, recall that a dual volume has been constructed
- Volume-based contributions are evaluated at the sub-control volume integration points (diamonds)
- Surface-based contributions are evaluated at the sub-control surface integration points (triangles)

We define an isoparametric element than ranges from -1:1 in the ξ– (x-direction) and η– (y-direction) direction



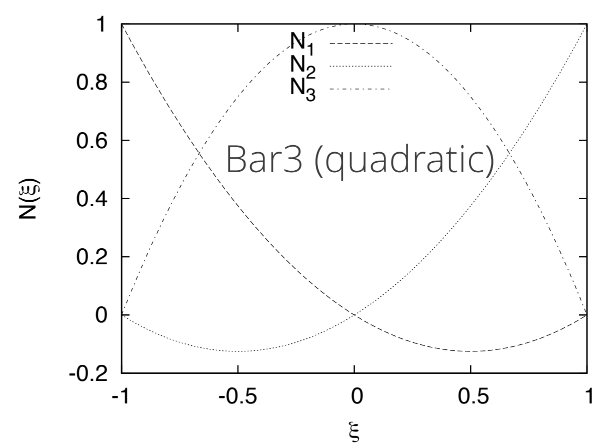
Basis Functions for a Quad4

$$N_1 = \frac{1}{4}(1-\xi)(1-\eta)$$

$$N_2 = \frac{1}{4}(1+\xi)(1-\eta)$$

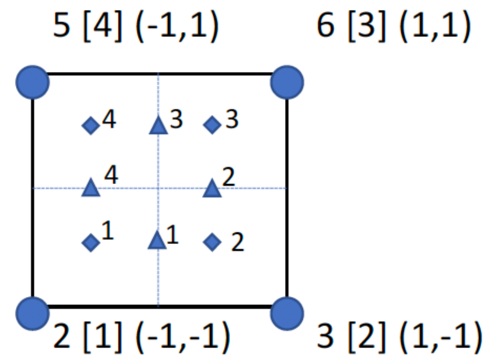
$$N_3 = \frac{1}{4}(1+\xi)(1+\eta)$$

$$N_4 = \frac{1}{4}(1-\xi)(1+\eta)$$



For example, the isoparametric coordinate for ip 1 is (0.0, -0.5)

Deep Dive on CVFEM: Implicit Time Discretization

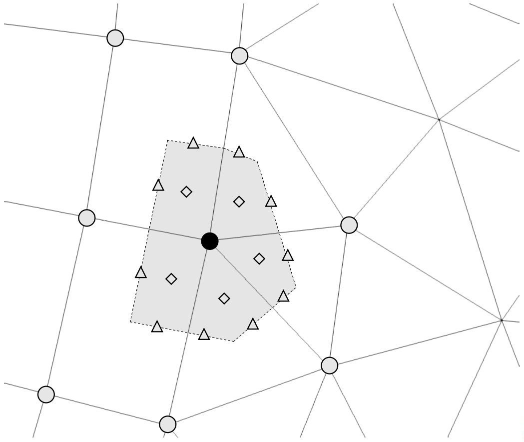


- Backward Euler (two state) and is first-order accurate (A-stable)
- BDF2 (three state) is second-order accurate (A-stable)
- This term is assembled over an element iteration and drives a consistent mass matrix with a full node:element:node connectivity

$$\int w \frac{\partial \rho \phi}{\partial t} dV \approx \sum_{ip} \frac{\left(\gamma_1 \rho_{ip}^{n+1} \phi_{ip}^{n+1} + \gamma_2 \rho_{ip}^{n} \phi_{ip}^{n} + \gamma_3 \rho_{ip}^{n-1} \phi_{ip}^{n-1}\right)}{\Delta t} V_{ip}$$

- For uniform time stepping, the coefficients are give by,
  - $\gamma_1 = 3/2$ $\gamma_2 = -2$ $\gamma_3 = ½$
- You can easily see how the underlying basis pulls in the full node:element_node stencil:

$$\phi_{ip} = \sum_{n} N_n^{ip} \phi_n$$



Deep Dive on CVFEM: Implicit Time Discretization; Code

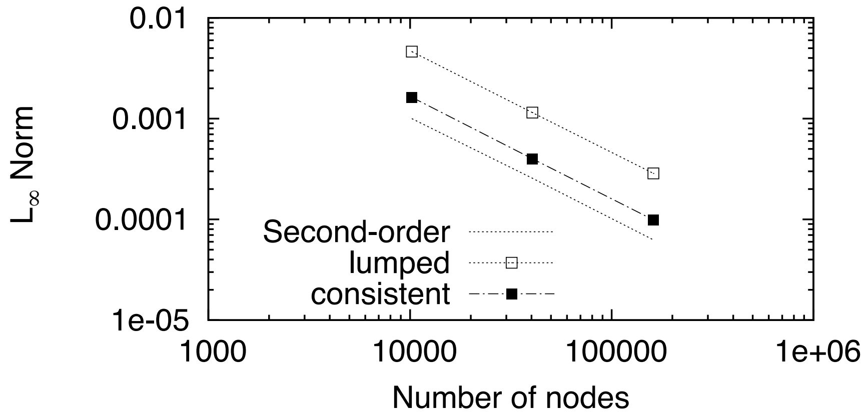
- https://github.com/NaluCFD/Nalu/blob/master/src/kernel/ScalarMassElemKernel.C

The Consistent Mass Matrix

- When evaluating the volumetric contributions using the full stencil, this approach is known as a *consistent mass matrix* as compared to the previous *lumped mass matrix* that is supported by CC and EBVC

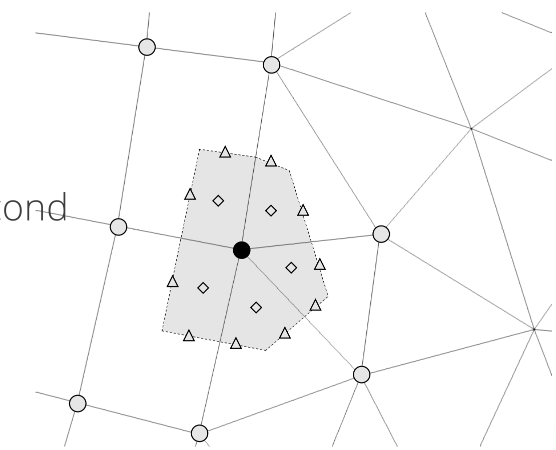
$$\int w \frac{\partial \rho \phi}{\partial t} dV \approx \sum_{ip} \frac{\left(\gamma_1 \rho_{ip}^{n+1} \phi_{ip}^{n+1} + \gamma_2 \rho_{ip}^{n} \phi_{ip}^{n} + \gamma_3 \rho_{ip}^{n-1} \phi_{ip}^{n-1}\right)}{\Delta t} V_{ip} \quad \text{(Consistent)} \qquad \phi_{ip} = \sum_n N_n^{ip} \phi_n$$

$$\int w \frac{\partial \rho \phi}{\partial t} dV \approx \sum_{nd} \frac{\left(\gamma_1 \rho_{nd}^{n+1} \phi_{nd}^{n+1} + \gamma_2 \rho_{nd}^{n} \phi_{nd}^{n} + \gamma_3 \rho_{nd}^{n-1} \phi_{nd}^{n-1}\right)}{\Delta t} V_{nd} \quad \text{(Lumped)}$$



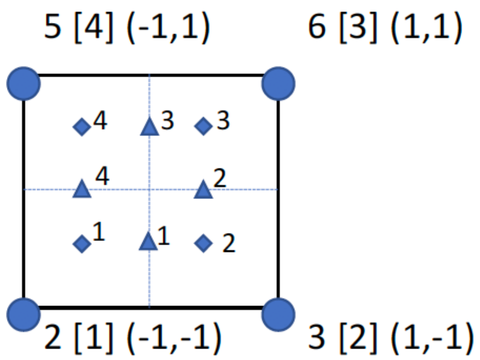
Convecting Taylor Vortex analytical solution; error computed at 1 second

Shown to be the convergence rate, 2nd order



Deep Dive on CVFEM: Source Term Discretization

- Source terms for CVFEM are also assembled over an element or nodal loop
- In some cases, the source term can be complex, i.e., includes gradients, which drives either a nodal assembly of these quantities to the nodes (recall, the projected nodal gradient), or local evaluation using the shape function derivatives (see upcoming diffusion operator)



$$\int wS^{\phi}dV \approx \sum_{ip} S^{\phi}_{ip}V_{ip}$$

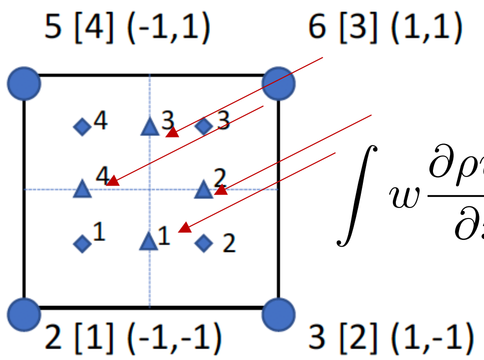
Deep Dive on CVFEM: Source Term Discretization; Code

- https://github.com/NaluCFD/Nalu/blob/master/src/user_functions/SteadyThermal3dContactSrcElemKernel.C

```
template<typename AlgTraits>
void
SteadyThermal3dContactSrcElemKernel<AlgTraits>::execute(
  SharedMemView<DoubleType**>& /* lhs */,
  SharedMemView<DoubleType *>& rhs,
  ScratchViews<DoubleType>& scratchViews)
{

  // Forcing nDim = 3 instead of using AlgTraits::nDim_ here to avoid compiler
  // warnings when this template is instantiated for 2-D topologies.
  NALU_ALIGNED DoubleType w_scvCoords[3];

  SharedMemView<DoubleType**>& v_coordinates = scratchViews.get_scratch_view_2D(*coordinates_);
  SharedMemView<DoubleType*>& v_scv_volume = scratchViews.get_me_views(CURRENT_COORDINATES).scv_volume;

  // interpolate to ips and evaluate source
  for ( int ip = 0; ip < AlgTraits::numScvIp_; ++ip ) {

    // nearest node to ip
    const int nearestNode = ipNodeMap_[ip];

    // zero out
    for ( int j =0; j < AlgTraits::nDim_; ++j )
      w_scvCoords[j] = 0.0;

    for ( int ic = 0; ic < AlgTraits::nodesPerElement_; ++ic ) {
      const DoubleType r = v_shape_function_(ip,ic);
      for ( int j = 0; j < AlgTraits::nDim_; ++j )
        w_scvCoords[j] += r*v_coordinates(ic,j);
    }

    rhs(nearestNode) += k_/4.0*(2.0*a_*pi_)*(2.0*a_*pi_)*(
      stk::math::cos(2.0*a_*pi_* w_scvCoords[0])
      + stk::math::cos(2.0*a_*pi_* w_scvCoords[1])
      + stk::math::cos(2.0*a_*pi_* w_scvCoords[2]))*v_scv_volume(ip);
  }
}
```

Deep Dive on CVFEM: Advection Discretization (no stabilization)

- For advection, we have transformed the volume integral to a surface integration
- Therefore, a patch of elements are required for the full assembly at node 2



$$\int w \frac{\partial \rho u_j \phi}{\partial x_j} dV \approx \sum_{ip} (\rho u_j)_{ip} \phi_{ip} n_j dS \approx \sum_{ip} \dot{m}_{ip} \phi_{ip}$$

Notes:

1. Common to integrate-by-parts, however, not required
2. Advection term need not be in divergence form (non-conserved form is suitable)

- Recall, that the mass flow rate at an integration point is prescribed
- Integration points can also be shifted from the sub-control surface to the edge midpoint (while still using the integration point area vector)
- This is a *central-* or *Galerkin-based* advection operator

Deep Dive on CVFEM: Advection Discretization Non-Conserved Form

- The advection term can be integrated by parts, or not; moreover, the PDE can drive a non-conservative equation form:

$$\int w \frac{\partial \rho u_j \phi}{\partial x_j} dV = \int w \rho u_j \phi n_j dS - \int \rho u_j \phi \frac{\partial w}{\partial x_j} dV$$

$$\int w \rho u_j \frac{\partial \phi}{\partial x_j} dV + \int w \phi \frac{\partial \rho u_j}{\partial x_j}$$

$$\int w \frac{\partial \rho \phi}{\partial t} dV = \int w \rho \frac{\partial \phi}{\partial t} dV + \int w \phi \frac{\partial \rho}{\partial t} dV$$

- Non-conserved form, unlike CC and EBVC, provides no added complexity

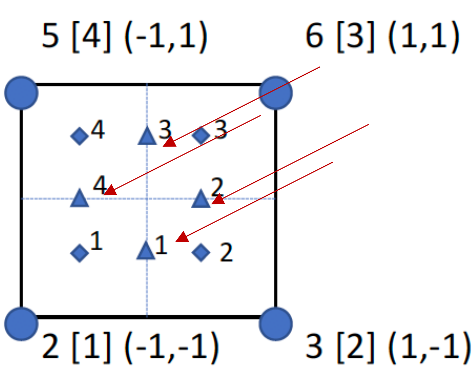
$$\int w \left(\rho \frac{\partial \phi}{\partial t} + \rho u_j \frac{\partial \phi}{\partial x_j} \right) dV$$

Deep Dive on CVFEM: Advection Discretization; Code

- https://github.com/NaluCFD/Nalu/blob/master/src/kernel/ScalarAdvDiffElemKernel.C
- This routine includes advection and diffusion
- Recall that integration point value is provided by nodal loop over the underlying nodal basis for this element
- Also note that this routine is valid for all types of supported elements – both low- and higher-order (polynomial promotion – coming later in the Quarter)

Deep Dive on CVFEM: Diffusion Discretization

- For diffusion, we have transformed the volume integral to a surface integration
- Therefore, a patch of elements are required for the full assembly at node 2
- Note that the CVFEM approach is absent any non-orthogonality corrections;
- However, high aspect ratio elements are now challenging…



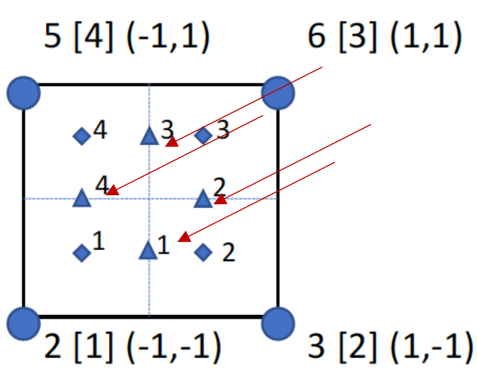
$$\int w \frac{\partial q_j}{\partial x_j} dV \approx -\sum_{ip} \frac{\mu}{Sc}_{ip} \frac{\partial \phi}{\partial x_j}_{ip} n_j dS = -\sum_{ip} \frac{\mu}{Sc}_{ip} \sum_{nd} \frac{\partial N_{nd}^{ip}}{\partial x_j} \phi_{nd} A_j^{ip}$$

Recall our underlying basis: $\phi_{ip} = \sum_n N_n^{ip} \phi_n$

$$\frac{\partial \phi}{\partial x_j}_{ip} = \sum_n \frac{\partial N_n^{ip}}{\partial x_j} \phi_n$$

Deep Dive on CVFEM: Diffusion Discretization

- For diffusion, we have transformed the volume integral to a surface integration
- Therefore, a patch of elements are required for the full assembly at node 2
- Note that the CVFEM approach is absent any non-orthogonality corrections;
- However, high aspect ratio elements are now challenging…

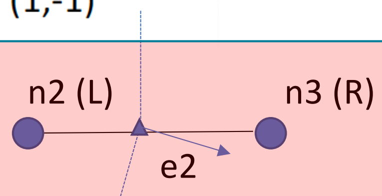


$$\int w \frac{\partial q_j}{\partial x_j} dV \approx -\sum_{ip} \frac{\mu}{Sc}_{ip} \frac{\partial \phi}{\partial x_j}_{ip} n_j dS = -\sum_{ip} \frac{\mu}{Sc}_{ip} \sum_{nd} \frac{\partial N^{ip}_{nd}}{\partial x_j} \phi_{nd} A^{ip}_j$$

Recall our underlying basis: $\phi_{ip} = \sum_n N_n^{ip} \phi_n$

$$\frac{\partial \phi}{\partial x_j}_{ip} = \sum_n \frac{\partial N_n^{ip}}{\partial x_j} \phi_n$$

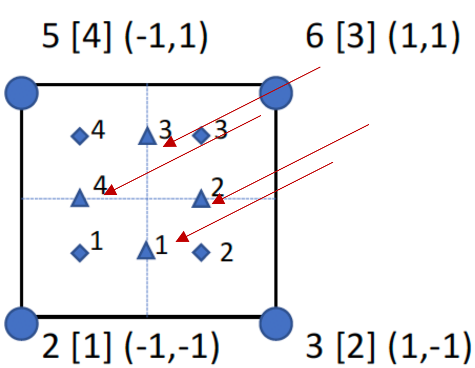
No non-orthogonality!



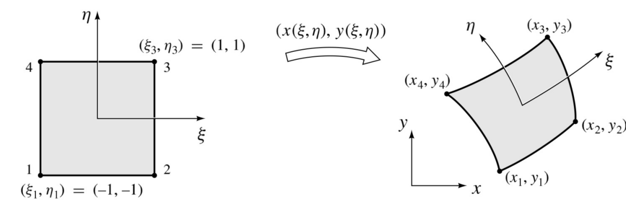
$$\frac{\partial \phi}{\partial x_j}|ip = G_j^{ip}\phi + \left[(\phi_R - \phi_L) - G_l^{ip}\phi \Delta x_l\right] \frac{A_j^{ip}}{A_k \Delta x_k}$$

Deep Dive on CVFEM: Diffusion Discretization

- For diffusion, we have transformed the volume integral to a surface integration
- Therefore, a patch of elements are required for the full assembly at node 2
- Note that the CVFEM approach is absent any non-orthogonality corrections;
- However, high aspect ratio elements are now challenging…



$$\begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \frac{\partial N}{\partial x} \\ \frac{\partial N}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial N}{\partial \xi} \\ \frac{\partial N}{\partial \eta} \end{bmatrix}$$

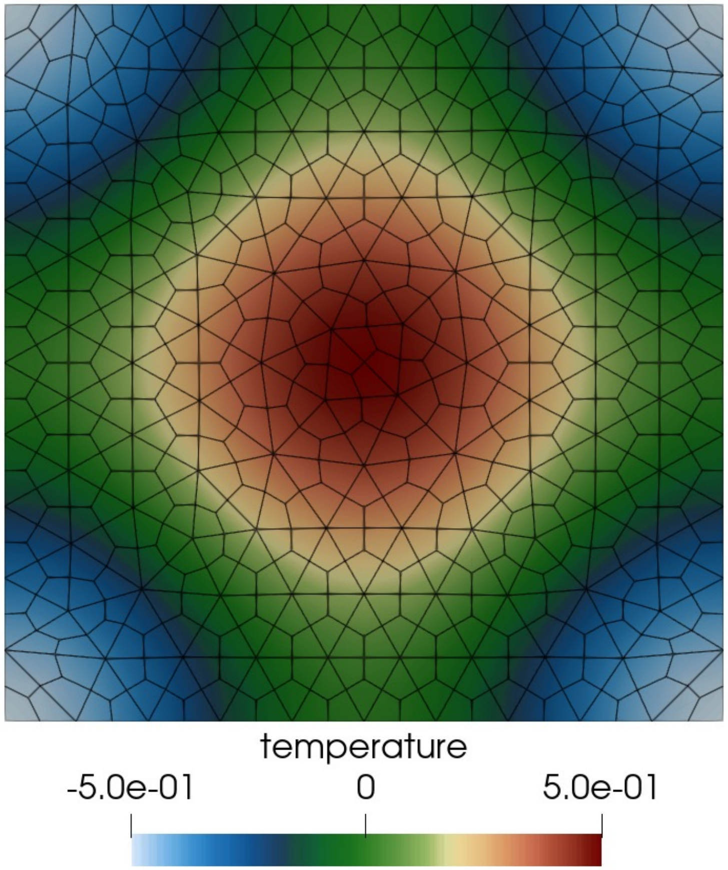


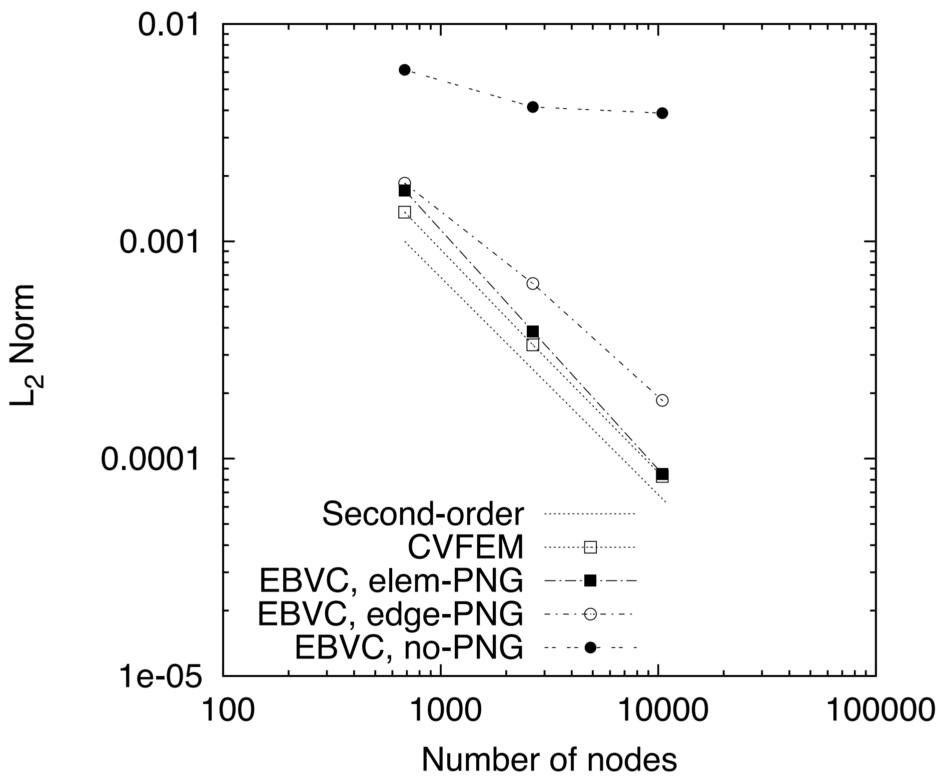
$$\begin{pmatrix} \frac{\partial N}{\partial x} \\ \frac{\partial N}{\partial y} \end{pmatrix} = J^{-1} \begin{pmatrix} \frac{\partial N}{\partial \xi} \\ \frac{\partial N}{\partial \eta} \end{pmatrix}$$

$$\mathbf{J} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{pmatrix}$$

$$\int w \frac{\partial q_j}{\partial x_j} dV \approx -\sum_{ip} \frac{\mu}{Sc}_{ip} \frac{\partial \phi}{\partial x_j}_{ip} n_j dS = -\sum_{ip} \frac{\mu}{Sc}_{ip} \sum_{nd} \frac{\partial N^{ip}_{nd}}{\partial x_j} \phi_{nd} A^{ip}_j$$

Deep Dive on CVFEM: Diffusion Discretization; Code





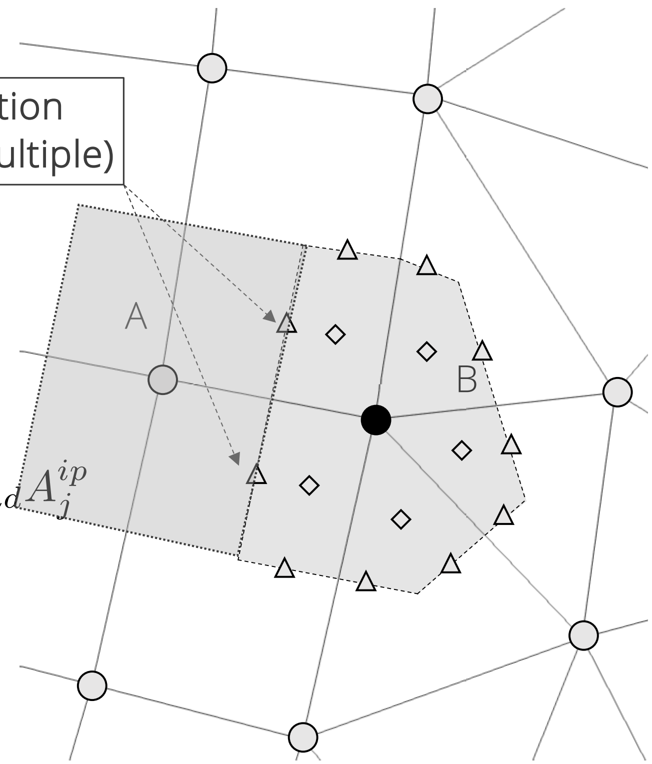
https://github.com/NaluCFD/Nalu/blob/master/src/kernel/ScalarAdvDiffElemKernel.C

A Note on Conservation

The CC, EBVC, and CVFEM all have one aspect in common...........

- Specifically, a flux contribution is evaluated at the control volume face, and dotted with the area surface normal vector
- This quantity, kg (stuff)/s, “leaves” control volume A, and fully enters control volume B and is *conserved*

$$\int w \frac{\partial q_j}{\partial x_j} dV \approx -\sum_{ip} \frac{\mu}{Sc}_{ip} \frac{\partial \phi}{\partial x_j}_{ip} n_j dS = -\sum_{ip} \frac{\mu}{Sc}_{ip} \sum_{nd} \frac{\partial N^{ip}_{nd}}{\partial x_j} \phi_{nd} A^{ip}_j$$



- For each integration point, the Left and Right nodal state is defined into which flux contributions are assembled
- No loss of mass/momentum/energy/etc.

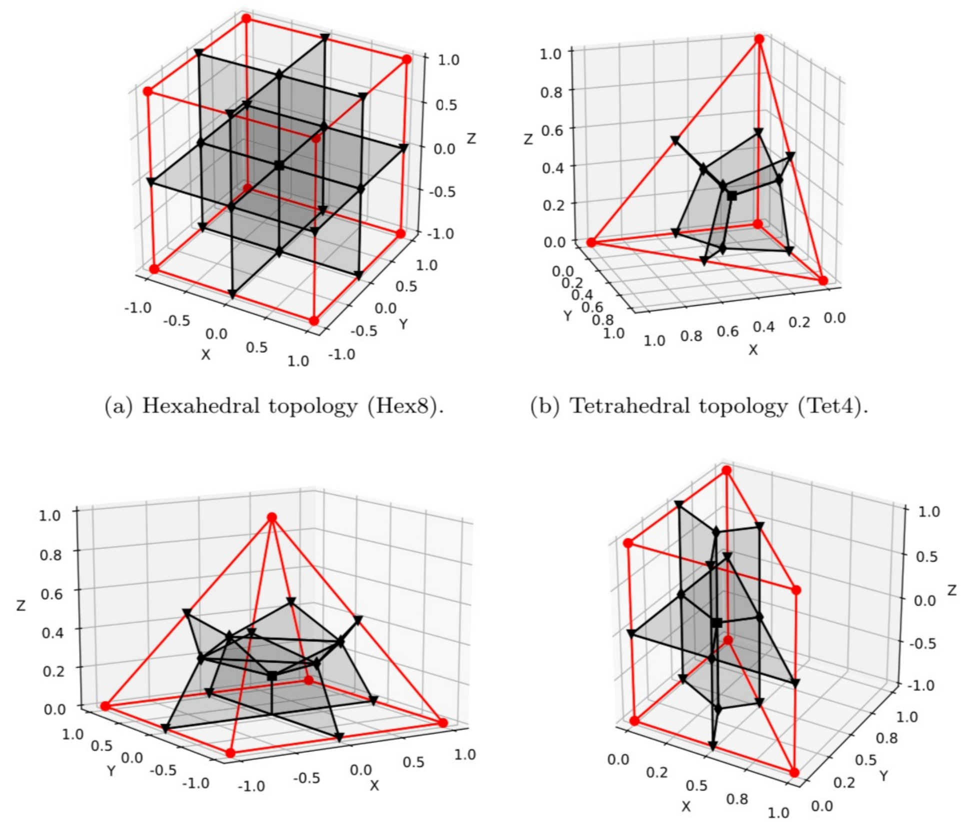
A Note on Polymorphism: Class Shape : Circle/Square/etc.

Recall, that the element type has been characterized by a set of unique attributes, take for example, a Hex8:

- Nodes Per Element: 8
- Number of surface integration points: 12
- Number of volume integration points: 8
- Number of Faces: 6
- Face Topology: Quad4
- Assembly Algoirthms can be templated, or expecting an integration rule defined by the element type

Pseudo C++

```
>Shape mySquare = new Square();
>Shape myCircle = new Circle();
>mySquare->volume();
>myCircle->volume();
```

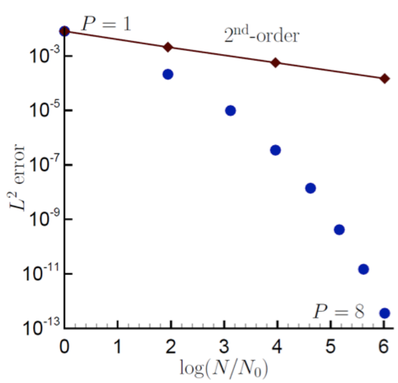


(a) Hexahedral topology (Hex8). (b) Tetrahedral topology (Tet4). (c) Pyramid topology (Pyramid5). (d) Wedge topology (Wedge6).

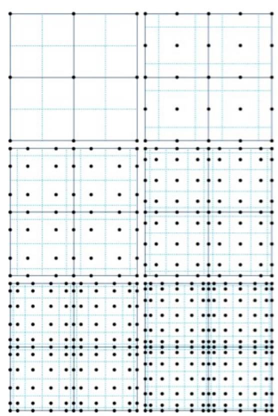
Fig. 1. CVFEM element and dual-volume definition for the low-order topologies.

Higher-Order via Polynomial Promotion

- CVFEM (like EBVC and CC) can be viewed as Petrov-Galerkin method
  - Recall how this is defined?
- Basis can also be promoted (linear to quadratic, etc), i.e., *Polynomial Promotion,* Domino, *CTRSP (*2014) as a first example of low-Mach fluids algorithm – or Domino, *JCP (*2018)
- Research Thrust: Possible higher efficiency on NGP due to increased local work)
- However, suitability of higher-order for LES is an open argument – especially when other errors/uncertainties exist

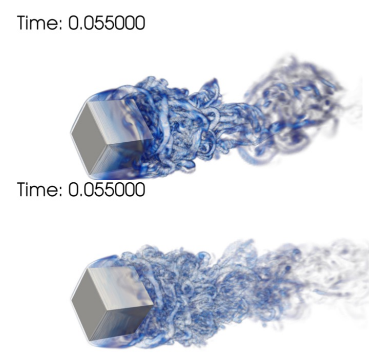


Spectral convergence



Dual-volume for promoted quad4

P=1 (left) and P=4 (right) Helium plume (VR-density)



Rotating cube (Re 4000, RPM 3600) P=1 (top) and P=2 (bottom)

CVFEM Review

- Finite volume
- Element-based
- Hybrid between finite element method and finite volume
- Underlying basis is tied to the element topology
- Operators allow for consistent integration at subcontrol surfaces and subcontrol volumes
- May be promoted in polynomial order
- Some advantages of operators in the presence of non-orthogonality
- Drives a more complex design in order to:
  - Manage multiple topologies, e.g., Hex, Tet, Wedge, Pyramid
  - Design computational kernels that can be re-used